How Computers See Pets: Build an Image Classifier with Python

Md. Izhar Ashraf

March 14, 2025

Contents

1 Introduction

1.1 What is Image Classification?

- **Definition:** Assigning labels to images (e.g., "cat" or "dog").
- **Example:** Facial recognition on smartphones, tumor detection in X-rays.
- **Analogy:** Like teaching a child to sort toys into boxes labeled "cars" and "dinosaurs".
- **Challenge for Machines:**
  - Computers see images as grids of numbers (pixels).
  - A cat image: A matrix like [[[255, 120, 0], ...]].

1.2 The Power of Neural Networks

- **Basic Idea:** Mimic the human brain's interconnected neurons.
- **How They Work:**
  - Input: Data (e.g., pixel values).
  - Layers: Process data through interconnected nodes (neurons).
  - Output: Prediction (e.g., "cat" or "dog").
  - "Learn" by adjusting connections (weights) to minimize errors.
- **Analogy:** Like a team of experts passing information, refining it at each step.

1.3 Why Convolutional Neural Networks (CNNs)?

- **Specialized for Images:** CNNs are a type of neural network designed for image data.
- **Human Vision Analogy:**
  - Step 1: Eyes detect edges (e.g., a cat's outline).
  - Step 2: Simplify and Focus; ignores noise (e.g., background grass).
  - Step 3: Organize Clues: combines edges into shapes (ears, tail).
  - Step 4: Recognizes the object: "That's a cat!"
- **How CNNs Mimic This:**
  - `Conv2D Layers`: Detect edges/textures (like cat fur vs. dog fur).
  - `Pooling Layers`: Simplify the image (focus on key regions).
  - `Flatten Layer`: Organize features into a checklist (e.g., "whiskers = 1, ears = 1").
  - `Dense Layers`: Decide if features match a "cat" or "dog".

1.4 Key Steps in Image Classification with CNNs

1. **Data Preparation:**
   - Organize images into folders (e.g., `base_directory/train/cats` and `/dogs`).
   - **Analogy:** Sorting laundry into "whites" and "colors" before washing.
2. **Preprocessing:**
   - Resize images to the same dimensions (e.g., 150x150 pixels).
   - Normalize pixel values from 0–255 to 0–1 (simplifies calculations).
   - Split data: 80% for training, 20% for validation.
   - **Analogy:** Preparing ingredients (chopping, measuring) before cooking.
3. **Building the CNN:**
   - `Conv2D Layers`: Detect edges/textures (like cat fur vs. dog fur).
   - `Pooling(Max/Av) Layers`: Simplify the image (focus on key regions).
   - `Flatten Layer`: Organize features into a checklist (e.g., "whiskers = 1, ears = 1").
   - `Dense Layers`: Decide if features match a "cat" or "dog".
4. **Training:**
   - **Epochs:** Number of times the model sees the entire dataset.
   - **Loss Function:** Measures how wrong the model's guesses are.
   - **Optimizer:** Adjusts the model's "brain" (weights) to reduce errors.
   - **Analogy:** A student learning from mistakes in practice exams.
5. **Prediction:**
   - Feed new images into the trained model.
   - Output: Probability (0–1) of being a dog (e.g., 0.2 = 20% dog $\rightarrow$ "cat").

1.5 Why Layers Matter in CNNs

- **Conv2D Layer 1:** Detects edges (horizontal/vertical lines).
- **Conv2D Layer 2:** Detects textures (fur, stripes).
- **Conv2D Layer 3:** Detects complex patterns (eyes, nose).
- **MaxPooling:** Makes the model robust to small shifts.

- **Analogy:** A detective solving a case:
  - Collect clues (edges) → Connect clues (textures) → Solve the case (identify cat/dog).

1.6 Key Steps in Image Classification

1. **Data Preparation:**
   - Organize images into folders (e.g., `base_directory/train/cats` and `/dogs`).
   - **Analogy:** Sorting laundry into "whites" and "colors" before washing.
2. **Preprocessing:**
   - Resize images to the same dimensions (e.g., 150x150 pixels).
   - Normalize pixel values from 0–255 to 0–1 (simplifies calculations).
   - Split data: 80% for training, 20% for validation.
   - **Analogy:** Preparing ingredients (chopping, measuring) before cooking.
3. **Building the CNN:**
   - `Conv2D + MaxPooling`: Repeatedly extract and simplify features.
   - `Flatten`: Convert 2D/3D features into a 1D list (e.g., turning a puzzle into a straight line of pieces).
   - `Dense Layers`: Connect all features to labels (cat/dog).
4. **Training:**
   - **Epochs:** Number of times the model sees the entire dataset.
   - **Loss Function:** Measures how wrong the model's guesses are (e.g., "You said cat, but it's a dog").
   - **Optimizer:** Adjusts the model's "brain" (weights) to reduce errors.
   - **Analogy:** A student learning from mistakes in practice exams.
5. **Prediction:**
   - Feed new images into the trained model.
   - Output: Probability (0–1) of being a dog (e.g., 0.2 = 20% dog → "cat").

1.7 Why Layers Matter in CNNs

- **Conv2D Layer 1:** Detects edges (horizontal/vertical lines).
- **Conv2D Layer 2:** Detects textures (fur, stripes).
- **Conv2D Layer 3:** Detects complex patterns (eyes, nose).
- **MaxPooling:** Makes the model robust to small shifts (e.g., cat facing left/right).
- **Analogy:** A detective solving a case:
  - Collect clues (edges) → Connect clues (textures) → Solve the case (identify cat/dog).

2 Hands-On Demo

2.1 Step 1: Setup and Data Loading

Listing 1: Import Libraries and Load Data

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import
    ↪ ImageDataGenerator

# Create data generators (rescale pixels to 0-1)
train_datagen = ImageDataGenerator(rescale=1./255,
    ↪ validation_split=0.2)

# Load images from folders: base_directory/train/cats and /dogs
train_generator = train_datagen.flow_from_directory(
    'base_directory/train',
    target_size=(150, 150),  # Resize all images to 150x150
    batch_size=32,
    class_mode='binary',      # 0 for cats, 1 for dogs
    subset='training')        # 80% for training

validation_generator = train_datagen.flow_from_directory(
    'base_directory/train',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='validation')      # 20% for validation
```

Key Points:

- Folders must be named `cats` and `dogs` inside `base_directory/train`.
- Resizing ensures all images have the same dimensions (like standardizing paper sizes).

2.2 Step 2: Build the CNN Model

Listing 2: Define CNN Architecture

```
model = tf.keras.Sequential([
    # Feature extraction layers
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
                           input_shape=(150, 150, 3)),  # 150
                              ↪ x150 RGB images
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    # Classification layers
    tf.keras.layers.Flatten(),  # Convert 3D features to 1D
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Output: 0
        ↪  (cat) or 1 (dog)
])

model.summary()  # Show model architecture
```

Analogy:

- `Conv2D + MaxPooling`: Like using a magnifying glass to find edges → then zooming out to see the bigger picture.
- `Flatten`: Organizing scattered puzzle pieces into a straight line.

2.3 Step 3: Train the Model

Listing 3: Compile and Train

```
model.compile(
    optimizer='adam',           # "Smart" weight adjustment
    loss='binary_crossentropy',# Measures prediction error
    metrics=['accuracy']        # Track % of correct guesses
)

# Train for 5 epochs (demo-friendly)
history = model.fit(
    train_generator,
    epochs=5,
    validation_data=validation_generator
)
```

Key Terms:

- **Epoch**: One full pass through the training data.
- **Accuracy**: "Score" for correct predictions (e.g., 90% = 9/10 correct).

2.4 Step 4: Live Prediction

Listing 4: Predict on New Images

```
from tensorflow.keras.preprocessing import image
import numpy as np

# Load a test image (e.g., test_cat.jpg)
img_path = 'test_cat.jpg'
img = image.load_img(img_path, target_size=(150, 150))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0) / 255.0  # Rescale pixels

prediction = model.predict(x)
print("Cat" if prediction[0] < 0.5 else "Dog")  # Threshold at
    ↪ 0.5
```